PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 17/30, 9/44 — A1

(11) International Publication Number: WO 99/33006

(43) International Publication Date: 1 July 1999 (01.07.99)

(21) International Application Number: PCT/US98/27247

(22) International Filing Date: 21 December 1998 (21.12.98)

(30) Priority Data:
60/068,415 22 December 1997 (22.12.97) US
09/106,189 29 June 1998 (29.06.98) US

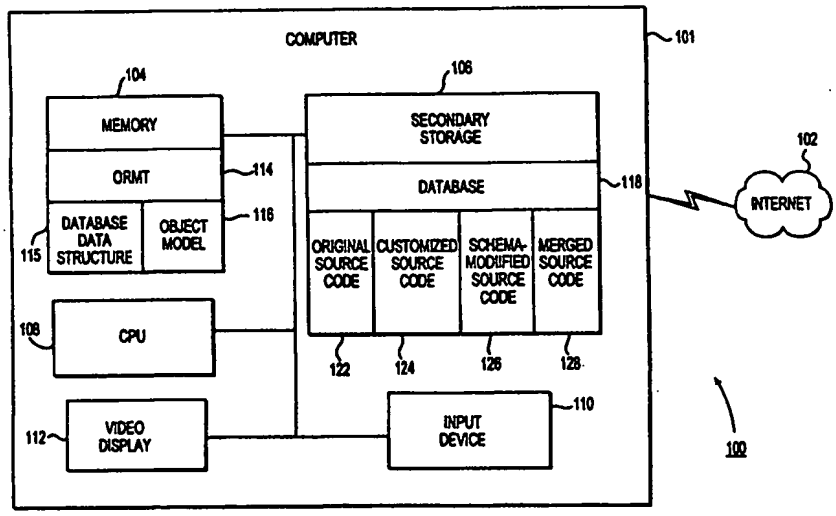
(71)(72) Applicants and Inventors: SHARMA, Rahul [IN/US]; Apartment #318, 3475 Granada Avenue, Santa Clara, CA 95051 (US). NG, Tony, Chun, Tung [CN/US]; 3716 Harlequin Terrace, Fremont, CA 94555 (US).

(74) Agents: GARRETT, Arthur, S. et al.; Finnegan, Henderson, Farabow, Garrett & Dunner, L.L.P., 1300 I Street, N.W., Washington, DC 20005–3315 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: EVOLUTION OF OBJECT–RELATIONAL MAPPING THROUGH SOURCE CODE MERGING



(57) Abstract

In accordance with methods and systems consistent with the present invention, an improved object–relational mapping tool is provided that merges two versions of source code: the first version reflects a database schema and contains customizations, and the second version reflects a modified database schema and does not contain the customizations. The merge results in source code containing classes which preserves both changes to the database schema as well as customizations to the source code. This functionality alleviates the programmer from having to recreate their customizations to the source code when the database schema changes, thus saving significant development time over conventional systems. Bef re the merge process is completed, the newly generated source code is displayed to the user with an indication of where each portion of the source code originates so that the user may manually override the merge process, thus providing the user with significant flexibility.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

EVOLUTION OF OBJECT-RELATIONAL MAPPING THROUGH SOURCE CODE MERGING

RELATED APPLICATIONS

The following identified U.S. patent applications are relied upon and are incorporated by reference in this application.

Provisional U.S. Patent Application No. 60/068,415, entitled System and Method for Mapping Between Objects and Databases," filed on December 22, 1997.

U.S. Patent Application No. _____, entitled "Object Relational Mapping Tool That Processes Views," bearing attorney docket no. 06502.0136-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Integrating Both Modifications to an Object Model and Modifications to a Database into Source Code by an Object-Relational Mapping Tool," bearing attorney docket no. 06502.0138-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Rule-Based Approach to Object-Relational Mapping Strategies," bearing attorney docket no. 06502.0139-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "User Interface for the Specification of Lock Groups," bearing attorney docket no. 06502.0142-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "A Fine-Grained Consistency Mechanism for Optimistic Concurrency Control Using Lock Groups," bearing attorney docket no. 06502.0143-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "User Interface for the Specification of Index Groups Over Classes," bearing attorney docket no. 06502.0144-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Method and Apparatus for Creating Indexes in a Relational Database Corresponding to Classes in an Object-Oriented Application," bearing attorney docket no. 06502.0145-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Method and Apparatus for Loading Stored Procedures in a Database Corresponding to Object-Oriented Data Dependencies," bearing attorney docket no. 06502.0146-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "An Integrated Graphical User Interface Method and Apparatus for Object-to-Database and Database-to-Object Mapping," bearing attorney docket no. 06502.0147-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Methods and Apparatus for Efficiently Splitting Query Execution Across Client and Server in an Object-Relational Mapping," bearing attorney docket no. 06502.0148-00000, and filed on the same date herewith.

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates generally to data processing systems and, more particularly, to merging source code in the context of an object-relational mapping tool.

Description of the Related Art

Object-relational mapping tools have been created to facilitate development of application programs that utilize a relational database. A relational database stores data in tables having rows (records) and columns (fields). The tables are usually interrelated, and thus, there is a logical structure imposed on the database. This logical structure is known as a schema.

The tables contained in a database are interrelated to each other by primary and foreign keys. Each table may have a primary key, comprising one or more columns, that uniquely identifies a row in the table. Foreign keys, on the other hand, are used to associate a row in one table with one or more rows in another table.

Object-relational mapping tools read a database and automatically generate source code from the database through a process known as mapping. This source code contains a number of classes whose interrelationships reflect the logical structure, or schema, of the database. A class, such as a Java™ class, is a data structure containing both data members

that store data and function members (or methods) that act upon the data. The source code contains one class for each table in the database, and each class has a data member for each column in the corresponding table. Additionally, the classes contain function members that are used to both get and set the values for the data members and, eventually, update the database.

By using an object-relational mapping tool, a programmer can automatically generate source code to facilitate their database application development. After the source code is generated, the programmer writes code to interact with only the classes in the source code and not the database, thus hiding the complexities of interacting with the database from the programmer. This allows a programmer who is familiar with object-oriented programming to code against familiar classes and not unfamiliar, sometimes cumbersome to use, database query languages.

Specifically, when a programmer runs the object-relational mapping tool, source code with classes is generated that reflects the structure (or schema) of the database at that time. Once a database schema has been mapped, it is common for the programmer to update and customize the classes in the source code (*e.g.*, change a field name, add or delete a field, or add comments). Likewise, during the lifetime of the database, it is also common for a database administrator to change the schema of the database (*e.g.*, add or delete a new field or table). As such, both the source code and the database tend to evolve and change over time.

Conventional object-relational mapping tools, however, are of little help in such situations. In order to ensure that the source code accurately reflects the database, the programmer must re-map the current database schema. However, the source code created by conventional mapping tools, in re-mapping the database, only reflects the subsequent modifications made to the database schema and does not reflect the customizations made to the source code by the programmer. Therefore, the programmer's previous customizations to the source code are lost and must be manually recreated, thus wasting significant development time. It is therefore desirable to improve object-relational mapping tools.

SUMMARY OF THE INVENTION

In accordance with methods and systems consistent with the present invention, an improved object-relational mapping tool is provided that merges two versions of source code:

the first version reflects a database schema and contains customizations, and the second reflects a modified database schema and does not contain the customizations. During the merge, the improved object-relational mapping tool examines the object models that corresponds to the source code files that are being merged. The examination of the source code's corresponding object model allows the improved object-relational mapping tool to make source code merge decisions based on the content of the corresponding object model. Thus, when merging a line of source code from one file with a line of source code from another file, the improved object-relational mapping tool determines how to merge the two lines based on the functionality performed by each line and the relationship of this line (e.g., a field) to other parts of the source code (e.g., the object containing the field). The improved object-relational mapping tool is able to determine the functionality performed by each line and the relationship of each line to other parts of the source code by examining the corresponding object model. By merging the source code based on content, the results are more accurate.

The merging of the two source code files results in source code containing classes which preserves both changes to the database schema as well as customizations to the source code. This functionality alleviates the programmer from having to recreate their customizations to the source code when the database schema changes, thus saving significant development time over conventional systems. Before the merge process is completed, the newly generated source code is displayed to the user with an indication of where each portion of the source code originates so that the user may manually override the merge process, thus providing the user with significant flexibility.

In accordance with methods consistent with the present invention, a method is provided in a computer system having a first source code file and a corresponding first object model reflecting a mapped database schema. This method creates a second source code file and a corresponding second object model reflecting a modified version of the database schema and compares the first source code file with the second source code file to isolate the modifications made so that a third source code file can be generated to reflect the modifications.

In accordance with methods consistent with the present invention, a method is provided in a computer system having a first source code file reflecting a database schema.

This method receives customizations into the first source code file, receives an indication that the database schema has been modified, generates a second source code file reflecting the modified database schema, and incorporates the customizations to the first source code and the modifications to the database schema into a third source code file.

In accordance with methods consistent with the present invention, a method is provided in a computer system having a first source code file reflecting a database schema and containing customizations and having a second source code file reflecting a modified version of the database schema. The method merges the first source code file and the second source code file to create a third source code file and displays the first, second, and third source code files to the user so that the user can selectively override the merging.

In accordance with systems consistent with the present invention, a data processing system is provided comprising a secondary storage device, a memory, and a processor. The secondary storage device contains a database having a logical structure comprising tables with rows and columns and a first source code file reflecting the logical structure of the database. The memory contains an object-relational mapping tool configured to operate after the logical structure of the database has been modified, configured to import the modified logical structure and create a second source code reflecting the modified logical structure, configured to create a third source code with the modifications, configured to display the first, second, and third source code files to the user, and configured to allow the user to selectively determine whether each modification should be incorporated into the third source code file. The processor is configured to run the object-relational mapping tool.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate an implementation of the invention and, together with the description, serve to explain the advantages and principles of the invention. In the drawings,

Figure 1 depicts a data processing system suitable for practicing methods and systems consistent with the present invention;

Figures 2A and 2B depict a flow chart of the steps performed to accomplish source code merging by an object-relational mapping tool in a data processing system depicted in Figure 1; and

Figures 3A and 3B depict a flow chart of the steps performed by the code-merging algorithm.

Figure 4 depicts a more detailed flow chart of the steps performed by the code merging algorithm depicted in Figures 3A and 3B when merging global level source code.

Figure 5 depicts a more detailed flow chart of the steps performed by the code merging algorithm depicted in Figures 3A and 3B when merging package level source code.

Figures 6A and 6B depict a more detailed flow chart of the steps performed by the code merging algorithm depicted in Figures 3A and 3B when merging class level source code.

Figures 7A and 7B depict a more detailed flow chart of the steps performed by the code merging algorithm depicted in Figures 3A and 3B when merging field level source code.

Figure 8 depicts a more detailed flow chart of the steps performed by the code merging algorithm depicted in Figures 3A and 3B when merging method level source code.

Figure 9 depicts a more detailed flow chart of the steps performed by the code merging algorithm depicted in Figures 3A and 3B when merging constructor level source code.

DETAILED DESCRIPTION OF THE INVENTION

Methods and systems consistent with the present invention provide an improved object-relational mapping tool that merges two versions of source code: the first version of the source code reflects a database schema and contains customizations received from a programmer, and the second version reflects a modified database schema and does not contain the customizations. Because the improved object-relational mapping tool merges these versions of source code, the programmer does not have to re-create their customizations to the source code when the database changes, thus saving significant development time over conventional systems.

During the merge, the improved object-relational mapping tool examines the object models that corresponds to the source code files that are being merged. The examination of the source code's corresponding object model allows the improved object-relational mapping tool to make source code merge decisions based on the content of the corresponding object model. Thus, when merging a line of source code from one file with a line of source code

from another file, the improved object-relational mapping tool determines how to merge the two lines based on the functionality performed by each line and the relationship of this line (e.g., a field) to other parts of the source code (e.g., the object containing the field). The improved object-relational mapping tool is able to determine the functionality performed by each line and the relationship of each line to other parts of the source code by examining the corresponding object model. By merging the source code based on content, the results are more accurate.

Overview

In accordance with methods and systems consistent with the present invention, the improved object-relational mapping tool maps a database by first querying the database to determine its schema and then by creating an internal data structure (known as the "database data structure") representing that schema. From this data structure, the object-relational mapping tool creates an object model containing all of the information necessary to generate classes and then creates source code containing a number of Java classes that may be used by a programmer to interface with the database. This source code is saved in a file and is referred to as the original source code. This mapping process is described in greater detail in co-pending U.S. Patent Application No. _____, entitled "An Integrated Graphical User Interface Method and Apparatus for Object-to-Database and Database-to-Object Mapping" which has previously been incorporated by reference.

At some point during the lifetime of the originally mapped source code, the programmer may want to evolve a class to reflect a real life business application (e.g., generate a report in some customized fashion). When the programmer customizes the original source code (*e.g.*, add, change, or delete fields), the customized version may be saved in the same or different file. The customized version of the original source code is referred to as the customized source code. It is possible for there to be many customized source code files, each potentially containing a different version of the programmer's changes.

Likewise, during the lifetime of the database, the requirements for the database evolve over time, thus compelling the database administrator (DBA) to make changes to the database (*e.g.*, add, change, or delete a table). After the database has been changed, the programmer may wish to update the customized code to reflect the database changes while maintaining the

previous customizations made to the source code. To accomplish this goal, the object-relational mapping tool, in accordance with methods and systems consistent with the present invention, imports the new database schema and creates a new version of the source code, referred to as the schema-modified source code.

The object-relational mapping tool then merges the customized source code with the schema-modified source code and creates a new version of the source code that contains both the customizations made to the source code and the modifications to the database. This new source code is referred to as the merged source code. The integration of the customized source code and the schema-modified source code is governed by a merge algorithm, referred to as the code-merging algorithm, which is described in further detail below.

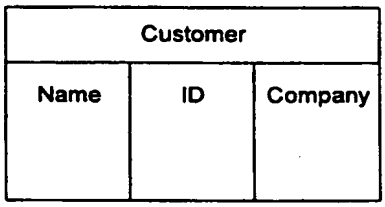
Specifically, the code-merge algorithm classifies each portion of the source code ("source code item"), such as a field or class, as either preserved, overridden, or inconsistent. This functionality is referred to as tagging (i.e., each item is tagged with this information). Items identified with the preserved tag will be saved in the merged source code file, while items identified with the overridden and inconsistent tags will be deleted from the merged source code file when the merge process is ultimately completed. The example provided below better explains how the tagging process works.

The object-relational mapping tool uses the code-merge algorithm to classify the source code items as preserved, overridden, or inconsistent. In general, the code-merge algorithm, based on various rules discussed in further detail below, classifies source code items that are in conflict with other items (e.g., one source code defines a class as having three fields, while the second source code defines the same field with four classes) as either preserved or overridden, respectively. Source code items involved in conflicts that the code-merge algorithm is unable to resolve (e.g., both source codes define the same field as something different) are classified as inconsistent, and require programmer assistance. Finally, source code items that are not involved in a conflict, during merging, in general, will be classified as preserved.

After merging, the object-relational mapping tool displays to the programmer, with the appropriate tag, the merged source code, which includes both the customizations made by the programmer and the modifications made by the database administrator. However, after the merged source code is displayed and before the merge process is completed, the

programmer may selectively edit the preserve, override, and inconsistent tags so as to affect the outcome of the merge process.

For instance, in the example shown below, a programmer may want to reverse a particular override classification (i.e., tag). In this example, the original source code reflects Table Customer by having three fields: Name, ID, and Company. Next to the database table is the database schema representing the database and, on the right, is the object based source code representing the database schema.



Database Schema

```
Table Customer:
    NAME        varchar (40),
    ID          int,
    COMPANY     varchar (40)
                }
```

Original Source Code

```
class Customer {
    private    String    name;
    private    int,      id;
    private    String    company
```

Next, the programmer customizes the original source code by changing the field named ID from an integer to a string and by adding a fourth field named address. Note, that the address field is a transient field because the field does not exist in the database schema (i.e., the Customer table).

Customized Source Code

```
class Customer {
        private  String            name;
        private  String            id;          // changed from int to string
        private  String            company;
        private  transient String  address       // transient field added
}
```

Likewise, the database administrator may modify the database by adding a phone number column and by adding an address column, the fourth and fifth columns respectively.

Customer				
Name	ID	Company	Phon	Address

The programmer then instructs the object-relational mapping tool to begin the merge process by re-importing the database schema.

Database Scheme

```
Table Customer:
    NAME        varchar (40),
    ID          int,
    COMPANY     varchar (40),
    PHONE       varchar (20),
    ADDRESS     varchar (100)

                }
```

Re-imported Schema Source Code

```
class Customer {
    private     String      name;
    private     int         id;
    private     String      company;
    private     String      phone;
    private     String      address
```

After an instruction to merge the customized source code with the re-imported modified schema source code is received by the mapping tool, the code-merge algorithm (CMA) merges the two files together and produces a merged source code file which is displayed to the programmer via a graphic user interface display window. The CMA, based on merging rules which are explained in greater detail below, determines whether source code will be preserved, overridden or tagged as inconsistent.

In the ongoing example, the CMA determines that the customized source code and the re-imported schema source code define the ID column differently (i.e., as a string and as an integer). According to the CMA, the re-imported schema source code defining the ID column will be tagged as preserved, while the customized source code will be overridden. In addition, the customized source code and the re-imported schema source code both label different fields by the same name (i.e., address). According to the CMA, the re-imported schema source code defining the field is tagged in the merged source code as preserved, while the customized source code is tagged in the merged source code as inconsistent. The tag inconsistent notifies the programmer that a merge conflict is present which requires programmer interaction. To resolve the inconsistency, however, the programmer can modify

the merged source code by changing the name (e.g., from address to address_zipcode) or by removing the field definition from the merged source code file. Nevertheless, if the programmer fails to correct the conflict, the CMA will discard the source code tagged as inconsistent in the final version of the source code. The remaining source code items, in this example, are classified as preserved since they are not in conflict. After generating the merged source code, it is displayed to the user. Note that when displayed to the user, the tags are color coded to better assist the programmer in customizing the merged source code file, which is depicted below.

Code Merging Source Code

```
                class Customer {
PRESERVED       private  String            name;
PRESERVED       private  int               id;          // override customized source code
OVERRIDDEN      private  String            id;
PRESERVED       private  String            company;
PRESERVED       private  String            phone;
PRESERVED       private  String            address;
INCONSISTENT    private  transient String  address
                                         }
```

After the programmer finishes editing the merged source code file, the object-relational mapping tool discards the source code that remains tagged as either overridden or inconsistent and saves the source code tagged as preserved. The results of the merge are stored in a different file, which can also be displayed to the programmer, thereby giving the programmer another opportunity to edit the source code. In this instance, the programmer resolved the inconsistency by renaming the customized source code associated with the transient field from address to address_zipcode.

Merged Source Code

```
                class Customer {
                        private     String            name;
                        private     int               id;
private         String              company;
                        private     String            phone;
                        private     String            address;
                        private     transient String  address_zipcode
                }
```

As a result, both the programmer's customizations to the original source code as well as the changes to the schema made by the database administrator are integrated into the

merged source code file, thus saving the programmer significant development time over conventional systems.

Implementations Details

Figure 1 depicts a data processing system 100 suitable for use with methods and systems consistent with the present invention. Data processing system 100 includes computer 101 connected to the Internet 102. Computer 101 includes memory 104, secondary storage device 106, central processing unit (CPU) 108, input device 110, and video display 112.

Memory 104 includes an object-relational mapping tool (ORMT) 114 in accordance with methods and systems consistent with the present invention. In turn, the object-relational mapping tool 114 includes object model 116 and database data structure 115, reflecting the schema of database 118, stored on secondary storage device 106. Also stored on secondary storage device 106 are a number of files, including original source code 122 reflecting the original schema of database 118, customized source code 124 reflecting the customizations made by a programmer to original source code 122, a schema-modified source code 126 reflecting the changes made by a database administrator to the database 118, and merged source code 128 reflecting the results from the automatic and manual merge process.

Although computer 101 is depicted with various components, one skilled in the art will appreciate that this computer can contain additional or different components. Additionally, although computer 101 is shown connected to the Internet 102, computer 101 may be connected to other networks, including other wide-area networks or local-area networks. Furthermore, although aspects of the present invention are described as being stored in memory and secondary storage, one skilled in the art will appreciate that these aspects can also be stored on or read from other types of computer-readable media, such as secondary storage devices, like hard disks, floppy disks, or CD-ROM; a carrier wave from a network like the Internet; or other forms of RAM or ROM. Sun, Sun Microsystems, the Sun logo, Java™, and Java™-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Figures 2A and 2B depict a flow chart of the steps performed during the source code merge process. First, a programmer instructs an object-relational mapping tool 114 to import a database schema 118 (step 202). In response to this request, the object-relational mapping

tool 114 creates a corresponding object model and a corresponding source code file 122 (step 204). This object model is referred to as the original object model and this source code is referred to as the original source code 122.

The steps performed by the object-relational mapping tool when generating an object model and a source code file from a database are described in greater detail in co-pending U.S. Patent Application No. _____, entitled "An Integrated Graphical User Interface Method and Apparatus for Object-to-Database and Database-to-Object Mapping" which has previously been incorporated by reference and in co-pending U.S. Patent Application No. _____, entitled "Integrating Both Modifications To An Object Model And Modifications To A Database Into Source Code By An Object-Relational Mapping Tool," which has also previously been incorporated by reference.

After original source code file 122 is generated, the programmer can create database application programs based on original source code file 122. These application programs, however, often entail customizing or altering the original source code file 122 (step 206). For example, the programmer, in creating an application, may add classes or redefine a particular field in a class. Once the programmer customizes the original source code file 122, the customizations can be saved either in a new source code file 124 or, simply, in the original source code file 122. In either case, the source code file is referred to as the customized source code file 124.

During the lifetime of the application program, the programmer stays apprised of whether the underlying database has been modified (e.g., by direct communication with the database administrator or by querying a relational database management system (RDBMS) which may manage the database) (step 210). The database schema can evolve, over time, for example by the addition or deletion of new tables or changes in the database tables by changing columns, constraints, or foreign keys.

If the database schema has not changed, then the programmer has no reason to activate the object-relational mapping tool since the customized source accurately reflects the database schema and processing ends. However, if the database schema did change, then the object-relational mapping tool imports the modified database schema (step 212). The object-relational mapping tool, when importing a database schema generates a new data structure

that represents an object model of the modified database and the corresponding schema-modified source code 126 (step 214).

After the object model and the source code has been generated for the modified database, the object-relational mapping tool generates an object model reflecting the customized source code 124 (state 215). Note, however, that if the programmer did not make any customizations to the original source code 122, then the object-relational mapping tool will not generate a new object model but, instead, will use the original object model for code merging purposes.

At this point, the object-relational mapping tool activates the code merging algorithm (CMA) which merges schema-modified source code file 126, which reflects the database administrator's database modifications, with either original source code file 122, which reflects the original database schema, or customized source code file 124, which reflects the programmer's customizations depending on whether the programmer made any customizations (step 216).

The object-relational mapping tool examines the two object models associated with the source code files that are being merged and based on pre-defined merger rules, explained in greater detail below, tags each source code item (e.g., field or class) as either preserved, overridden or inconsistent. This is referred to as tagging (i.e., each item is tagged with this information). Source code items identified with the overridden and inconsistent tags will be deleted from the merged source code file, while source code items identified with the preserved tag will be saved, when the merge process is ultimately completed.

The object-relational mapping tool stores the schema modified source code and the customized source code including the corresponding tags, into a new file, referred to as the merged source code file. This file, via a graphic-user interface, is displayed to the programmer (step 218). In addition, each displayed item contains reference to its respective source. This can be done by using colors to represent the respective source code files.

After the merged source code is displayed but before the merge process is completed, the programmer determines whether or not to customize the merged source code file by selectively editing the preserve and override determinations of the object-relational mapping tool that occurred during the code merging process (step 220 in Figure 2B). This edit feature allows the user to selectively accept or reject the automated source code merge decisions on a

case-by-case basis and thereby customize the resulting merged source code file. If the programmer decides to manually edit the merged source code file 128, then the programmer makes the appropriate edits by using the graphic-user interface of the object-relational mapping tool (step 222).

Independent of whether the programmer chooses to manually override the decisions made by the code merge algorithm, the programmer determines if there are any source code items that are classified as inconsistent. If there are any source code items that are tagged as inconsistent, the programmer determines whether or not to edit the source code associated with the inconsistent tag (step 224). If the programmer chooses to edit the inconsistent source code (step 226), the programmer must make sufficient changes to assuage the inconsistency (e.g., change the name of an inconsistent field), otherwise the object-relational mapping tool will generate an error during the compile of the merged source code file.

Independent of whether the programmer engages the editor to overcome inconsistencies, the next step is for the object-relational mapping tool to discard the source code that remains tagged as either overridden or inconsistent from the merged source code file 128 (step 227). As a result, both the programmer's customizations to the original source code file as well as the changes to the database schema made by the database administrator are integrated into the merged source code file.

Once the merged source code file 128 contains only the source code that was tagged as preserved, the object-relational mapping tool allows the programmer to view the file, via the display screen (step 228). At this point, the programmer can either continue editing the merged source code file 128 or, if no editing occurs, processing ends.

Code Merge Algorithm

Figures 3A and 3B depict a flow chart of the steps performed by the code-merge algorithm (CMA). The object-relational mapping tool compares, on six different levels, the schema-modified source code with the customized source code. Then, based on a set of pre-defined rules that are based on the functionality performed by each item of source code, as discussed below, the object-relational mapping tool will either tag each source code item as preserved, overridden, or inconsistent. For the purposes of this disclosure, a "source code item" is any source code statement, including fields, methods, classes or portions thereof.

The object-relational mapping tool determines the functionality of each source code item by examining the source code item's corresponding data structure represented by the object model. Object models and how they are compared are described in greater detail in co-pending U.S. Patent Application No. _____, entitled "An Integrated Graphical User Interface Method and Apparatus for Object-to-Database and Database-to-Object Mapping" and U.S. Patent Application No. ____, entitled "Integrating Both Modifications to an Object Model and Modifications to a Database into Source Code by an Object-Relational Mapping Tool," which have previously been incorporated by reference.

More specifically, the object-relational mapping tool performs code merging on a file level and requires an enhanced Java parser with an abstract syntax tree (AST) tailored for merging needs and can recognize all mergable programming constructs in the Java™ Programming Language (e.g., fields, methods, constructors) and is able to tag the mergable constructs with additional information (e.g., line numbers, whether changed or not). The parser is a mechanism to separate the source code into programming constructs and AST is a data structure that relates the individual constructs to each other and to the underlying classes in a hierarchical manner.

The first step by the CMA is to examine the global level source code and determine whether or not there are any source code conflicts on the global level between the customized source code 124 and the schema-modified source code 126 (step 302 in Figure 3A). Source code on the global level includes package statements and user comments. The package statement at the global level is the list of the related classes that are defined in the underlying database. The concept of a "package" and other concepts related to the Java programming language are described in greater detail in Gosling, Joy, and Steele, The Java Language Specification, Addison-Wesley (1996), which is incorporated herewith by reference. If there are any conflicts in the source code at the global level between the customized source code and the schema modified source code, the CMA initiates global level source code merging (step 304). Global level source code merging is depicted in Figure 4 and is explained in greater detail below. After the CMA completes the merging process for the conflicting global level source code items or if there are no conflicts at the global level, the CMA will include one version of the remaining like or similar global level source code items and tag the source code as preserved (step 306).

Next, the CMA examines the package level source code and determines whether or not there are any source code conflicts on the package level between the customized source code 124 and the schema-modified source code 126 (step 308). Source code on the package level includes the package definition, which includes additional related classes added by the programmer, import statements, which import the classes defined in the package statement, accessibility source code, which defines access privilege, and user comments. If there are any conflicts in the source code at the package level between the customized source code and the schema modified source code, the CMA initiates package level source code merging (step 310). Package level source code merging is depicted in Figure 5 and is explained in greater detail below. After the CMA completes the merging process for the conflicting package level source code items or if there are no conflicts at the package level, the CMA will include one version of the remaining like package level source code items and tag the source code as preserved (step 312).

Next, the CMA examines the class (i.e., table or view) level source code and determines whether or not there are any source code conflicts on the class level between the customized source code 124 and the schema-modified source code 126 (step 314). Source code on the class level includes the addition and deletion of classes (i.e., tables or views), class name changes, inner classes (i.e., classes defined within another class), inheritance (i.e., a way to determine if there is a subclass relationship between classes), relationships, accessibility source code, and user comments. If there are any conflicts in the source code at the class level between the customized source code and the schema modified source code, the CMA initiates class level source code merging (step 316). Class level source code merging is depicted in Figures 6A and 6B and is explained in greater detail below. After the CMA completes the merging process for the conflicting class level source code items or if there are no conflicts at the class level, the CMA will include one version of the remaining like or similar class level source code items and tag the source code as preserved (step 318).

After class level merging, the CMA examines the field (i.e., column) level source code and determines whether or not there are any source code conflicts on the field level between the customized source code 124 and the schema-modified source code 126 (step 320 in Figure 3B). Source code on the class level includes the addition and deletion of columns and fields, field and column name changes, column type, accessibility and user comments. If

there are any conflicts in the source code at the field level between the customized source code and the schema modified source code, the CMA initiates field level source code merging (step 322). Field level source code merging is depicted in Figures 7A and 7B and is explained in greater detail below. After the CMA completes the merging process for the conflicting field level source code items or if there are no conflicts at the field level, the CMA will include one version of the remaining like field level source code items and tag the source code as preserved (step 324).

Next, the CMA examines the method (i.e., get and set methods) level source code and determines whether or not there are any source code conflicts on the method level between the customized source code 124 and the schema-modified source code 126 (step 326). Source code on the method level includes the addition and deletion of methods, accessibility, and user comments. If there are any conflicts in the source code at the method level between the customized source code and the schema modified source code, the CMA initiates method level source code merging (step 328). Method level source code merging is depicted in Figure 8 and is explained in greater detail below. After the CMA completes the merging process for the conflicting method level source code items or if there are no conflicts at the method level, the CMA will include one version of the remaining like class level source code items and tag the source code as preserved (step 330).

Finally, the CMA examines the constructor (i.e., initialization methods) level source code and determines whether or not there are any source code conflicts on the constructor level between the customized source code 124 and the schema-modified source code 126 (step 332). Source code on the constructor level includes the addition and deletion of constructors, accessibility, and user comments. If there are any conflicts in the source code at the constructor level between the customized source code and the schema modified source code, the CMA initiates constructor level source code merging (step 334). Constructor level source code merging is depicted in Figure 9 and is explained in greater detail below. After the CMA completes the merging process for the conflicting method level source code items or if there are no conflicts at the method level, the CMA will include one version of the remaining like class level source code items and tag the source code as preserved (step 336).

At this point, there will be a merged source code file that contains the results from the CMA. Each source code item will be tagged as either preserved, overridden, or inconsistent. Processing returns to 218 in Figure 2A.

Global Level Source Code Merging

The first step for merging source code at the global level is to determine if the package statement for the schema-modified source code is in conflict with the package statement for the customized source code (step 402 in Figure 4). For example, there would be a conflict if the packages statements contained different names. If the package statements is in conflict, then the CMA tags the package statement source code representing the schema-modified source code file as preserved (step 404) and tags the package statement source code representing the customized source code as overridden (step 406).

Next, the CMA determines if there is any user comments added by the programmer at the global level (step 410). If there are user comments at the global level, then the CMA tags user comment source code representing the customized source code as preserved (step 412). After tagging the user comments or if there were no user comments added by the programmer, processing ends.

Package Level Source Code Merging

The first step for merging source code at the package level is to determine if the package definition in the customized source code contains an additional class (step 502 in Figure 5). If the package definition contains an additional class, the CMA determines if the class name for the additional class is unique (step 504). Each class name in the source code file must be unique. If the additional class has a unique name (i.e., does not conflict with a class name in the package definition for the schema-modified source code, then the CMA tags the package definition source code as preserved (step 506). However, if the additional class has a name that is being used by the class definition for the schema-modified source code, then the CMA tags the package definition source code as inconsistent (step 508).

Next the CMA determines if the import statement source code for the customized source code was changed by the programmer (step 510). If the import statement was

changed, then the CMA tags the import statement source code for the customized source code as preserved (step 512).

If the import statement for the customized source code was tagged or if the programmer made no changes to the import statement, the CMA next determines whether the accessibility scope is in conflict (step 516). The accessibility scope defines whether a particular class is private, protected or public. If the accessibility scope of a class is in conflict, the CMA determines whether or not the conflict will make the class reflecting the schema-modified source code less restrictive (step 518). For example, if the class originally was public (i.e., accessible to all classes regardless of whether the other classes are subclasses from this package) and the programmer customized the class to be protected (i.e., accessible only to the class and its subclasses) and then the database administrator changed the class to private (i.e., accessible only to the class itself), upon a merge, the customization would be in conflict since the customization would allow restricted access while the database manager would not allow any access.

Accordingly, if the CMA determines the accessibility customization is in conflict, i.e., the customization is less restrictive than the database schema, the customization is tagged as preserved and the schema modified source code is tagged as overridden (step 620). Otherwise, if the customization is not in conflict, i.e., the customization is more restrictive or the same, then the customization is tagged as overridden and the schema modified source code is tagged as preserved (step 522).

Independent of whether the accessibility is in conflict on the package level, the CMA next determines if the programmer customized the source code at the package level by creating user comments (step 524). Any creation of user comments by the programmer are tagged as preserved (step 526). Independent of whether the programmer customized the source code by adding comments, processing ends.

Class Level Source Code Merging

The first step for merging source code at the class level is to determine if the database administrator changed the name of a table or view (step 602 in Figure 6A). A view is simply a subset of one or more tables. For example, a view may be defined as two of the columns from a seven column table. Tables and views, when imported by an object-relational

mapping tool, correspond to a class. A newly imported class has the same name as its corresponding table or view in the underlying database. Both the programmer and the DBA can change the name of these tables and views. If the DBA changed the name of a table or view, then the schema-modified source code representing the class with the changed name is tagged as preserved and any customized source code that is in conflict with newly named class is tagged as overridden (step 610).

If the CMA tagged the class or if there was no change in name, then the CMA next determines if a table or view was added by the DBA (step 608). If there is a new table or view, then the CMA tags the source code from the schema-modified source code file that is associated with the new class as preserved (step 604). Note, that if the new class is in conflict with a class from the customized source code, the customized source code class is tagged as overridden.

If the CMA tagged the class or if no table or view was added by the DBA, then the CMA determines if a table or view was deleted by the DBA (step 614). If a table or view was deleted by the DBA, then the CMA tags the source code from the customized source code file that is associated with the deleted class as overridden (step 616).

Independent of whether a table or view is deleted, the CMA next determines whether the programmer customized the source code by creating inner (or nested) classes (step 620). Inner classes are subclasses defined within another class, and can be nested on any level. Inner classes do some useful logic or event handling for the defined class or associated methods. The addition of source code by the programmer representing inner classes is tagged as preserved (step 622).

Whether or not the programmer added inner classes, the CMA next determines if the programmer added a new class (624). If the customized source code contains a new class, the CMA determines if the name of the new class is unique (step 626). If the name is unique, then CMA tags the source code that represents the new class as preserved, since there are no conflicts (step 628). However, if the class name is not unique, i.e., the name also represents a table or view in the schema-modified source code, then the CMA tags the source code that represents the new class as inconsistent and tags the source code that represents the conflicting table or view from the schema-modified source code as preserved (step 630).

After the CMA tags the new class source code or after the CMA determines that there was no new class added by the programmer, the CMA next determines whether any inheritances were created or changed in the mapped object model due to a change or an addition of a table or view by the database administrator (step 632 in Figure 6B). Inheritance is a way to determine if there is a subclass relationship between classes. For example, when the primary key of table Red is also a foreign key that references the primary key of a class Blue, then class Blue inherits from class Red. If an inheritance is created, then the CMA tags the respective class with the inheritance information and tags the corresponding source code as preserved (step 634). In addition, if the new inheritance relationships via the schema-modified source code conflict with the customized source code, the CMA tags the customized source code that is in conflict as overridden.

Independent of whether there is an inheritance created by a new table or view, the CMA determines whether any relationships were created by the addition of a table (or view) to the database by the database administrator (step 636). A relationship is a way to describe whether or not a table (or view) is related to another table (or view) via foreign keys. If a relationship has been created, then the CMA tags the source code corresponding to the respective class with the relationship information as preserved (step 638). In addition, if the new relationship information via the schema-modified source code conflict with the customized source code, the CMA tags the source code from the customized source code that is in conflict as overridden.

Independent of whether a relationship was added due to database changes by the DBA, the CMA next determines, whether the accessibility scope is in conflict (step 640). The accessibility scope defines whether a particular class is private, protected or public. If the accessibility scope of a class is in conflict, the CMA determines whether or not the conflict will make the class reflecting the schema-modified source code less restrictive (step 642). Accordingly, if the CMA determines the accessibility customization is in conflict, i.e., the customization is less restrictive than the database schema, the customization is tagged as preserved and the schema modified source code is tagged as overridden (step 646). Otherwise, if the customization is not in conflict, i.e., the customization is more restrictive or the same, then the customization is tagged as overridden and the schema modified source code is tagged as preserved (step 644).

Independent of whether the database administrator changed the accessibility of a class, the CMA next determines if the programmer customized the source code at the class level by creating user comments (step 648). Any creation of user comments by the programmer are tagged as preserved (step 650). Independent of whether the programmer customized the source code by adding comments, processing ends.

Field Level Source Code Merging

The first step for merging source code at the field level is to determine whether the name of a field in the mapped object model had been changed or deleted due to a change in the name of a column in the database table or view (step 702 in Figure 7A). If a field name was changed or deleted by the DBA, the CMA tags the customized source code that corresponds to the field as inconsistent and tags the schema-modified code that corresponds to the changed field as preserved (step 704). If DBA actually deleted the field, then the CMA simply honors the delete in the schema-modified source code.

Independent of if a column name was changed by the DBA, the CMA next determines whether the field type in the mapped object model had been changed due to a change in the column type in a table or view (step 708). If a column type was changed, then the CMA tags the customized source code corresponding to the field type as inconsistent and tags the schema-modified source code corresponding to the column as preserved. (step 704).

Next step by the CMA, independent of whether the DBA changed the column type, is to determine if a column was added by the DBA (step 714). If a field has been added, the CMA determines whether the name of the column is unique (step 716). If the name of the field is unique, then the CMA tags the schema-modified field source code corresponding to the new column and methods to both get and set the value of this field as preserved (step 718). However, if the new column name is not unique, then the CMA tags the schema-modified source code as preserved and tags the customization source code as inconsistent (step 720).

Independent of whether the DBA added a field, the CMA next determines if the programmer customized the source code by adding a transient field (step 722). A transient field is a field that is added by the programmer to an existing class. It is transient in the sense that it does not exist in the database schema. Transient fields can be contrasted with

persistent fields in that persistent fields do exist in the database schema. If a transient field has been added by the programmer, the CMA determines if the name of the transient field is unique (step 724). If the transient field name is unique, the CMA tags the corresponding source code for the transient field as preserved (step 726). However, if the transient field name is not unique, the CMA tags the customized source code corresponding to the transient field as inconsistent and the tags the schema-modified source code with the similar name as preserved (step 728).

Independent of whether the programmer customized the source code by adding transient fields, the CMA next determines if the programmer customized the source code by deleting a persistent field (step 730 in Figure 7B). A persistent field is a field that exists in the database schema. If a persistent field is deleted by the programmer, the CMA determines whether the schema-modified source code still reflects that particular field (step 731). If the schema-modified source code reflects that particular field then the CMA will tag the source code for the field as preserved and include the appropriate methods (e.g., get and set methods) (step 732). If the schema-modified source code does not reflect that particular field then the CMA does not tag any source code since neither source code file utilizes the field.

Independent of whether the programmer deleted a persistent field, the CMA next determines whether the accessibility scope is different (step 736). The accessibility scope defines whether a particular field is private, protected or public. If the accessibility scope of a field is in conflict, the CMA determines whether or not the conflict will make the field reflecting the schema-modified source code less restrictive (step 738). Accordingly, if the CMA determines the accessibility customization is in conflict, i.e., the customization is less restrictive than the database schema, the customization is tagged as preserved and the schema-modified source code is tagged as overridden (step 740). Otherwise, if the customization is not in conflict, i.e., the customization is more restrictive or the same, then the customization is tagged as overridden and the schema modified source code is tagged as preserved (step 742).

Independent of whether the database administrator changed the accessibility of a field, the CMA next determines if the programmer customized the source code at the field level by creating user comments (step 744). Any creation of user comments by the programmer are

tagged as preserved (step 746). Independent of whether the programmer customized the source code by adding comments, processing ends.

Method Level Source Code Merging

The first step for merging source code at the method level is for the CMA to determine whether the programmer customized the source code by adding a method (step 802 in Figure 8). If the programmer adds a method, then customized source code corresponding to the method, added by the programmer, is tagged by the CMA as preserved (step 804).

Independent of whether the programmer customized the source code by adding a method, the CMA next determines if the programmer customized the source code by deleting an accessor method (i.e., get methods) or a modifier method (i.e., set methods) (step 806). If a method was deleted by the programmer, the CMA honors the deletion by tagging the schema-modified source code that represents the methods that were deleted as overridden (step 810).

Independent of whether the programmer customized the source code by deleting accessor or modifier methods, the CMA next determines whether the accessibility scope is different (step 812). If the accessibility scope of a method is not the same in the two source code files, the CMA determines whether or not the conflict will make the method reflecting the schema-modified source code less restrictive (step 814). Accordingly, if the CMA determines the accessibility customization is in conflict, i.e., the customization is less restrictive than the database schema, the customization is tagged as preserved and the schema-modified source code is tagged as overridden (step 816). Otherwise, if the customization is not in conflict, i.e., the customization is more restrictive or the same, then the customization is tagged as overridden and the schema modified source code is tagged as preserved (step 818).

Independent of whether the database administrator changed the accessibility of a method, the CMA next determines if the programmer customized the source code at the method level by creating user comments (step 820). Any creation of user comments by the programmer are tagged as preserved (step 822). Independent of whether the programmer customized the source code by adding comments, processing ends.

Constructor Level Source Code Merging

Next, the CMA determines if the programmer customized the source code by adding a constructors (step 902 in Figure 9). A constructor is a method used to create an object or a method used to initialize fields with default values. If the programmer adds a constructor, the customized source code that corresponds to the constructor is tagged as preserved (step 904).

Independent of whether the programmer customized the source code by adding constructors, the CMA next determines if the programmer customized the source code by deleting a constructor (step 906). If the programmer deletes a constructor, the schema-modified source code that corresponds to the deleted constructor is tagged as preserved (step 908). This has the effect of overriding the programmer's decision to delete the constructor.

Independent of whether the programmer customized the source code by deleting constructors, the CMA next determines whether the accessibility scope is different (step 910). If the accessibility scope of a constructor is not the same in the two source code files, the CMA determines whether or not the conflict will make the constructor reflecting the schema-modified source code less restrictive (step 912). Accordingly, if the CMA determines the accessibility customization is in conflict, i.e., the customization is less restrictive than the database schema, the customization is tagged as preserved and the schema-modified source code is tagged as overridden (step 914). Otherwise, if the customization is not in conflict, i.e., the customization is more restrictive or the same, then the customization is tagged as overridden and the schema modified source code is tagged as preserved (step 916).

Independent of whether the database administrator changed the accessibility of a constructor, the CMA next determines if the programmer customized the source code at the constructor level by creating user comments (step 918). Any creation of user comments by the programmer are tagged as preserved (step 920). Independent of whether the programmer customized the source code by adding comments, processing ends.

Conclusion

In accordance with the present invention an improved object-relational mapping tool has been described that is able to merge a programmer's source-code customizations with a database administrator's changes to the underlying database schema. It accomplishes this goal by creating source code for the changed database schema and comparing the source code

with the source code that corresponds to the programmer's customizations. The resulting merged source code is then displayed to the programmer who has the option of manually and selectively overriding the algorithm based merge decisions and manually editing any inconsistencies that may arise during the merge process. After this manual merge process, the merged source code replaces the previous customized version.

Although methods and systems consistent with the present invention have been described with reference to an exemplary embodiment thereof, those skilled in the art will know of various changes in form and detail which may be made without departing from the spirit and scope of the present invention as defined in the appended claims and their full scope of equivalents.

WHAT IS CLAIMED IS:

1. A method in a data processing system with a database having a schema and an object-relational mapping tool, the method performed by the object-relational mapping tool comprising:

generating first source code with first code items reflecting the schema of the database;

receiving customizations into the first source code;

receiving modifications into the schema of the database;

generating second source code with second code items reflecting the modified schema;

merging the first source code with the second source code to generate third source code by:

examining one of the first code items and one of the second code items;

determining, based on the examination, whether the one first code item supersedes the one second code item;

copying both the one first code item and the one second code item into the third source code;

tagging the one first code item as preserved and the one second code item as overridden in the third source code when it is determined that the one first code item supersedes the second source code item; and

tagging the one second code item as preserved and the one first code item as overridden in the third source code when it is determined that the one first code item does not supersede the second source code item;

displaying contents of the third source code to a user;

receiving user modifications to the tags in the third source code; and

generating fourth source code containing a portion of the contents of the third source code tagged as preserved.

2. A method in a data processing system having a first source code with first code items and a second source code with second code items, comprising:

examining the first code items and the second code times; and

merging the first code items and the second code items based on a predefined set of rules.

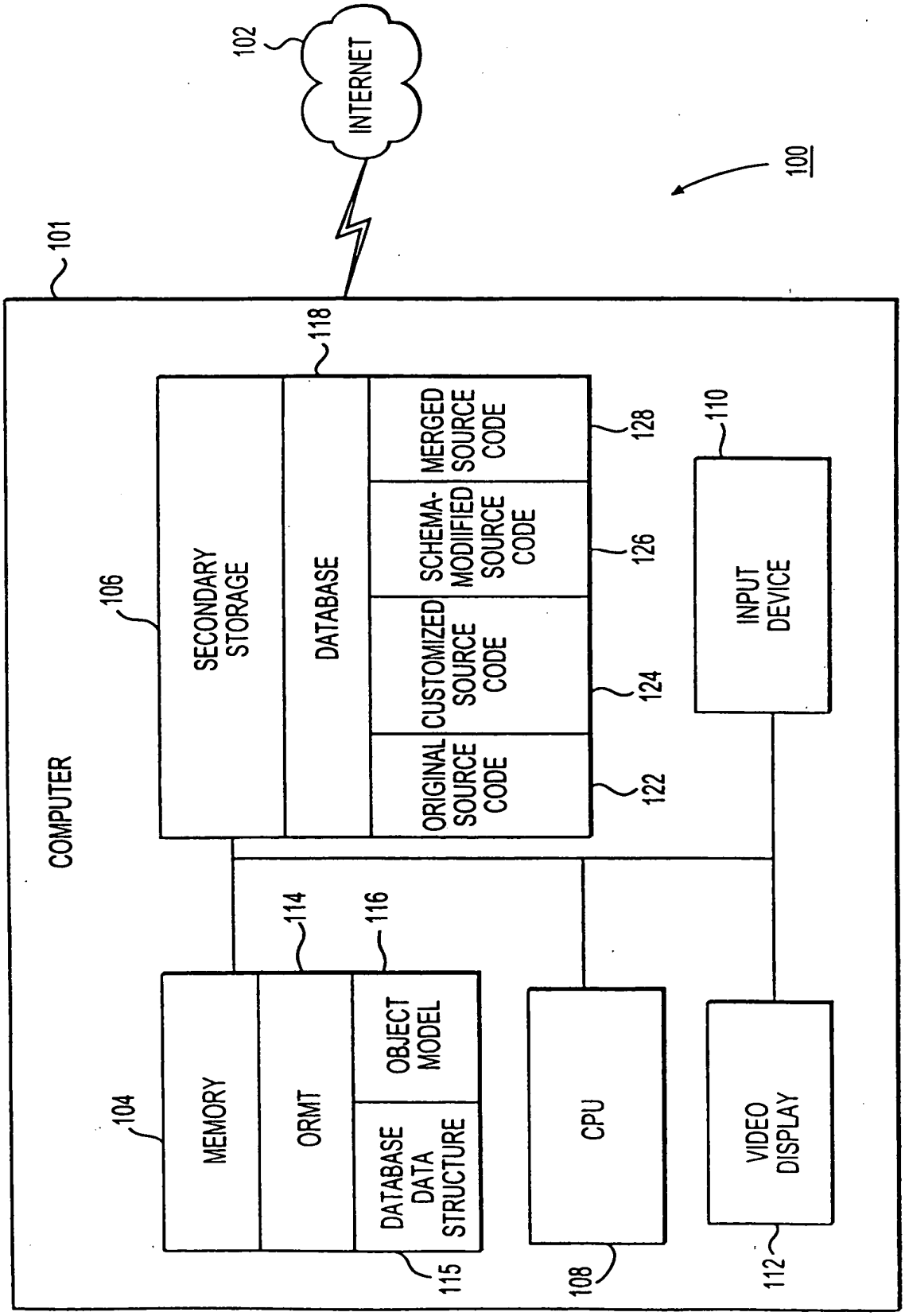


FIG. 1

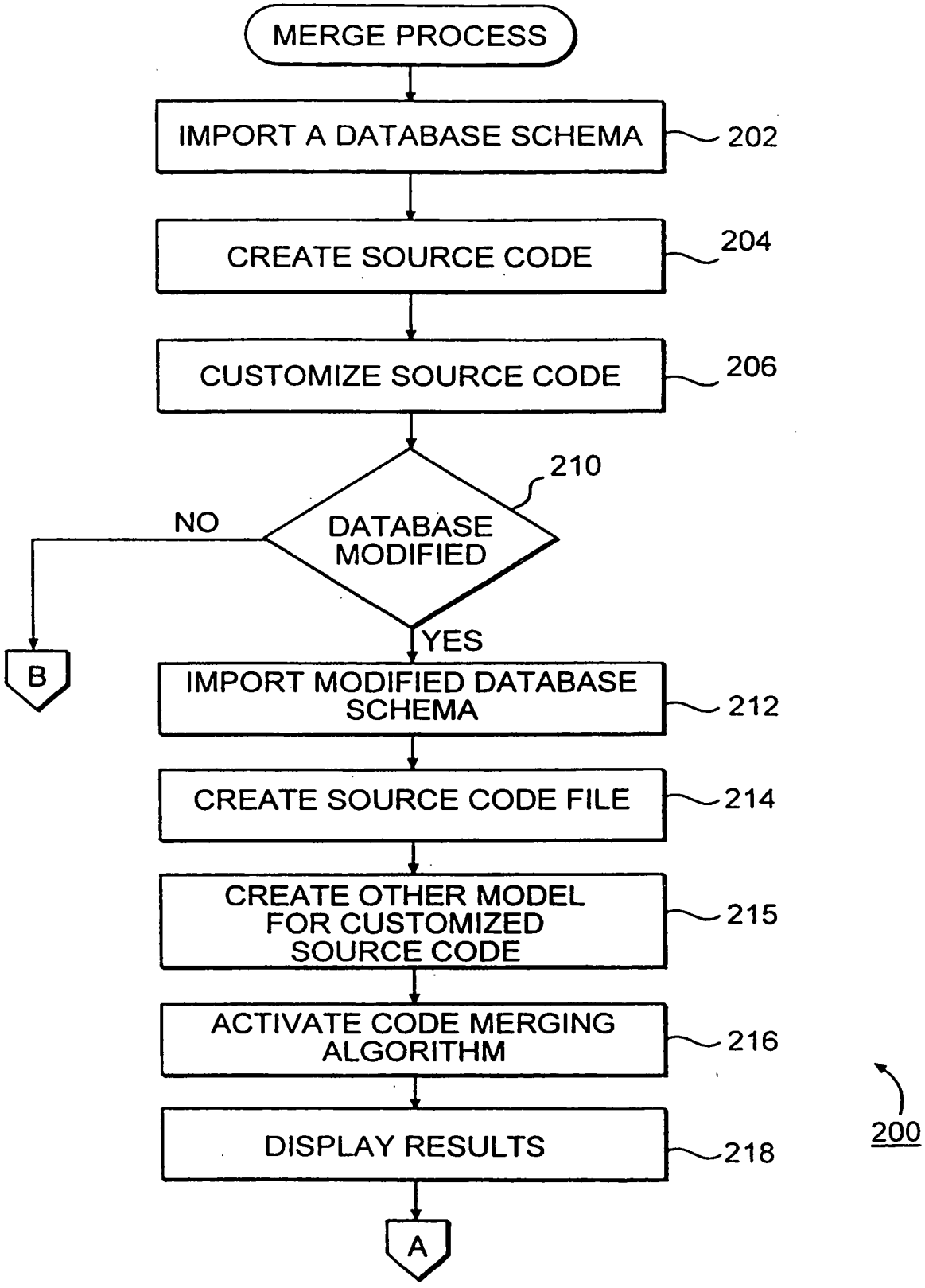


FIG. 2A

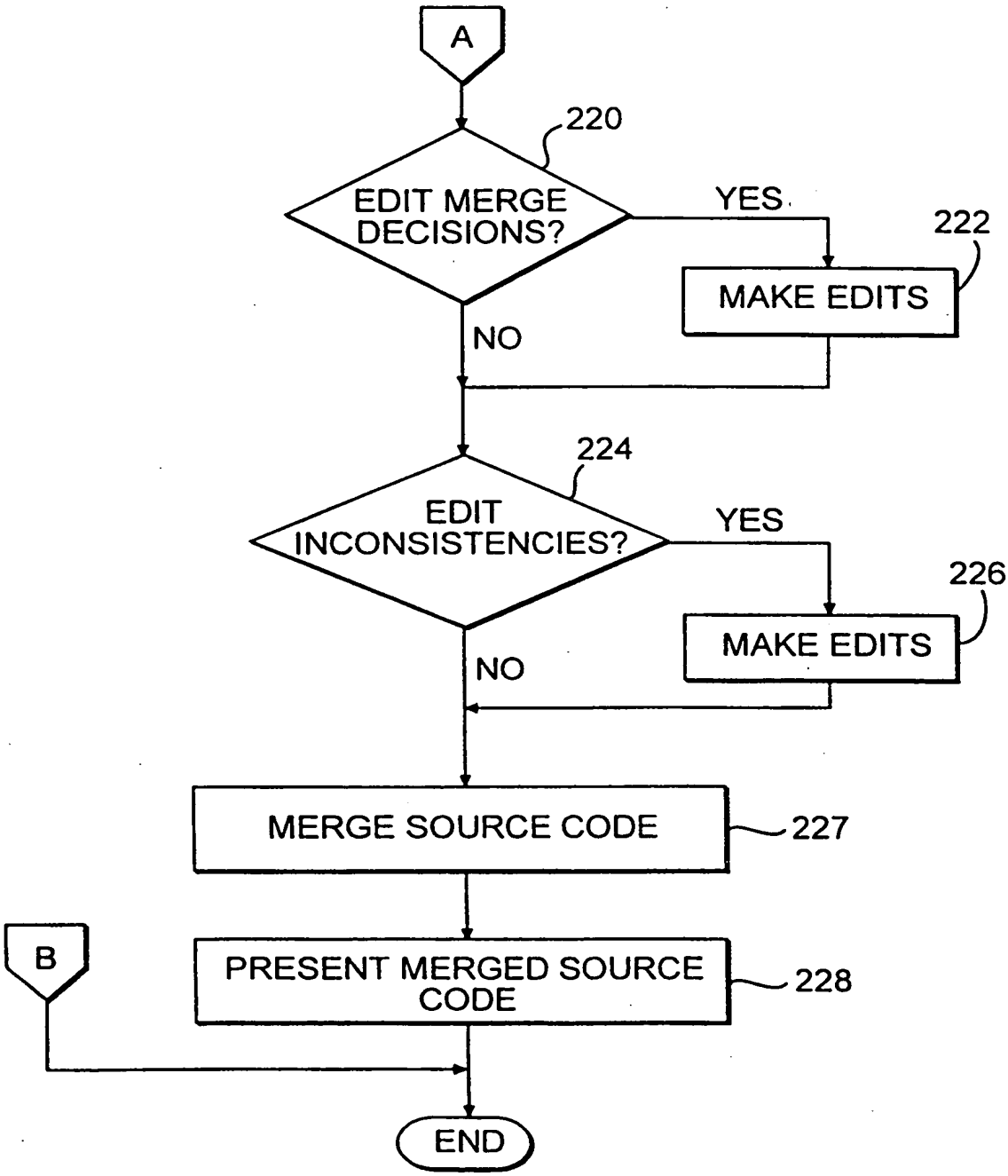


FIG. 2B

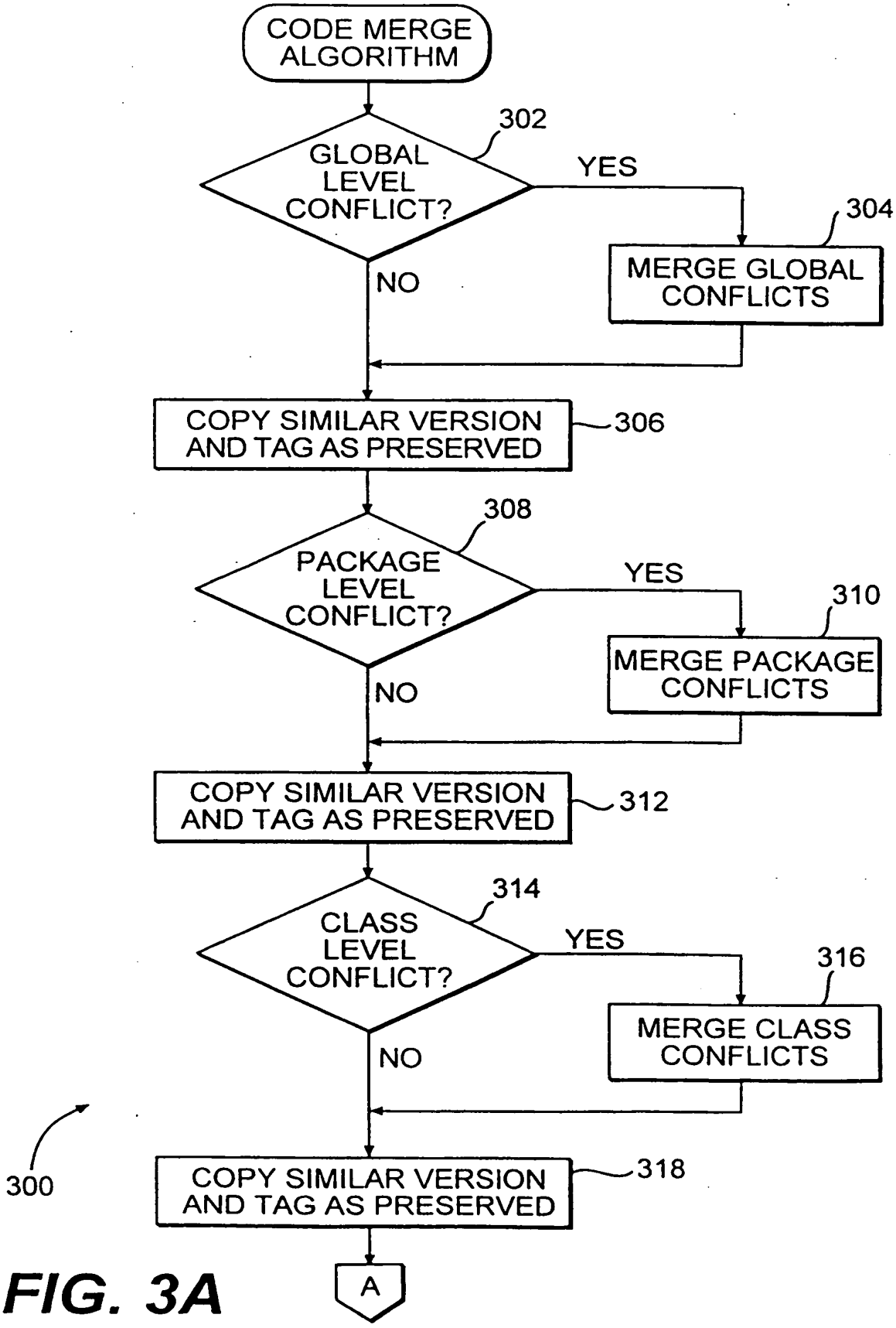


FIG. 3A

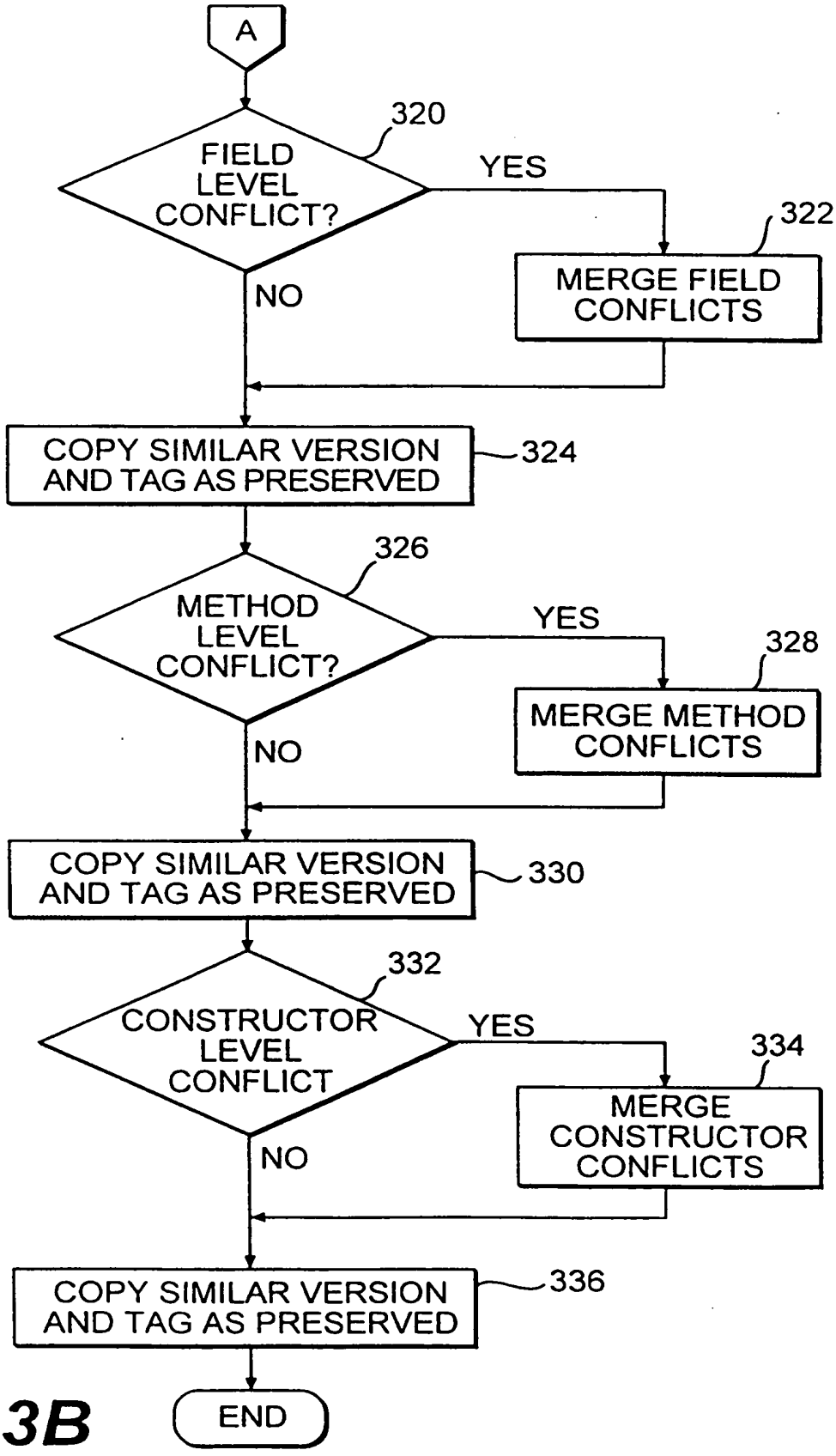

A
320
FIELD LEVEL CONFLICT?
YES
322
MERGE FIELD CONFLICTS
NO
COPY SIMILAR VERSION AND TAG AS PRESERVED
324
326
METHOD LEVEL CONFLICT?
YES
328
MERGE METHOD CONFLICTS
NO
COPY SIMILAR VERSION AND TAG AS PRESERVED
330
332
CONSTRUCTOR LEVEL CONFLICT
YES
334
MERGE CONSTRUCTOR CONFLICTS
NO
COPY SIMILAR VERSION AND TAG AS PRESERVED
336
END

FIG. 3B

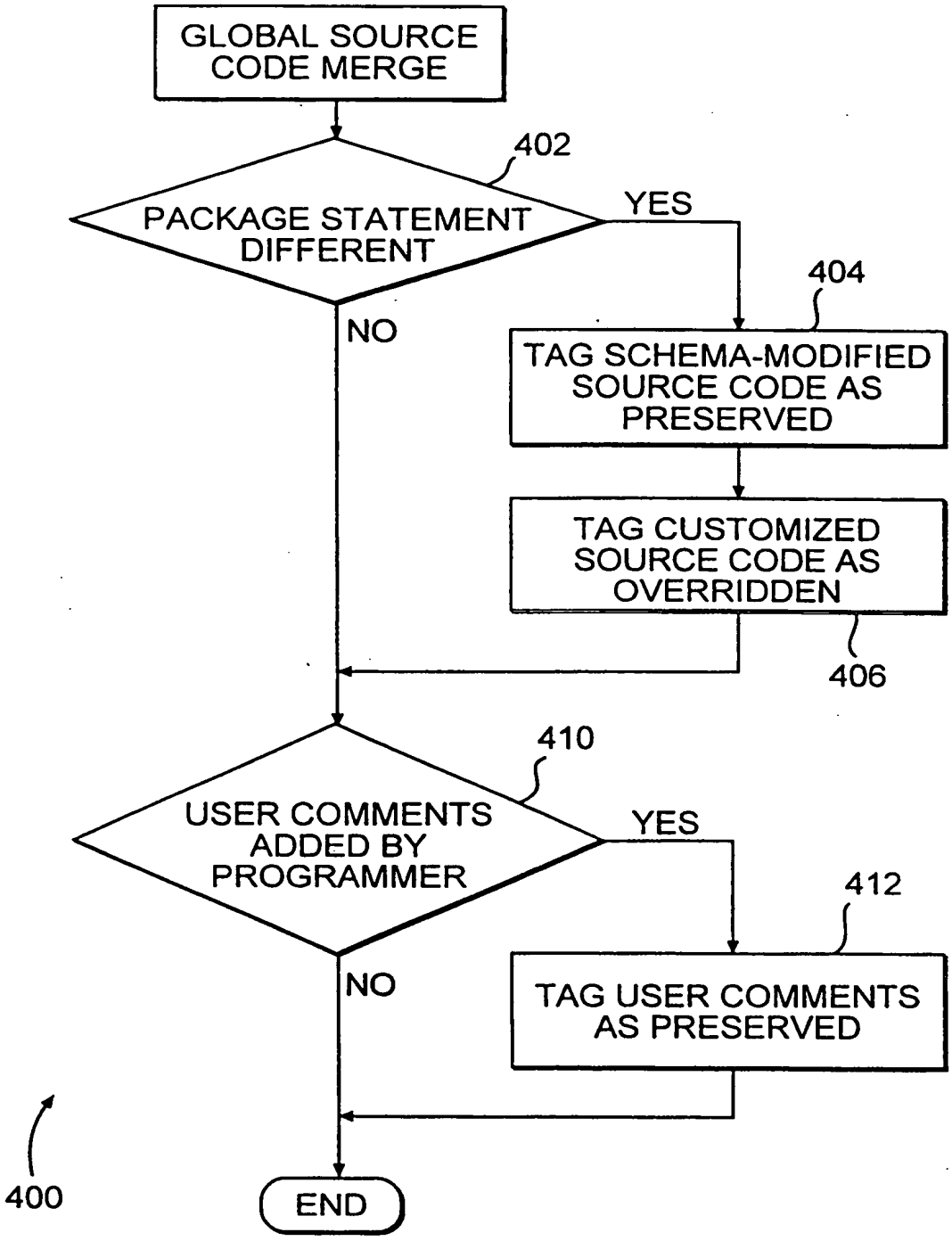


FIG. 4

FIG. 5

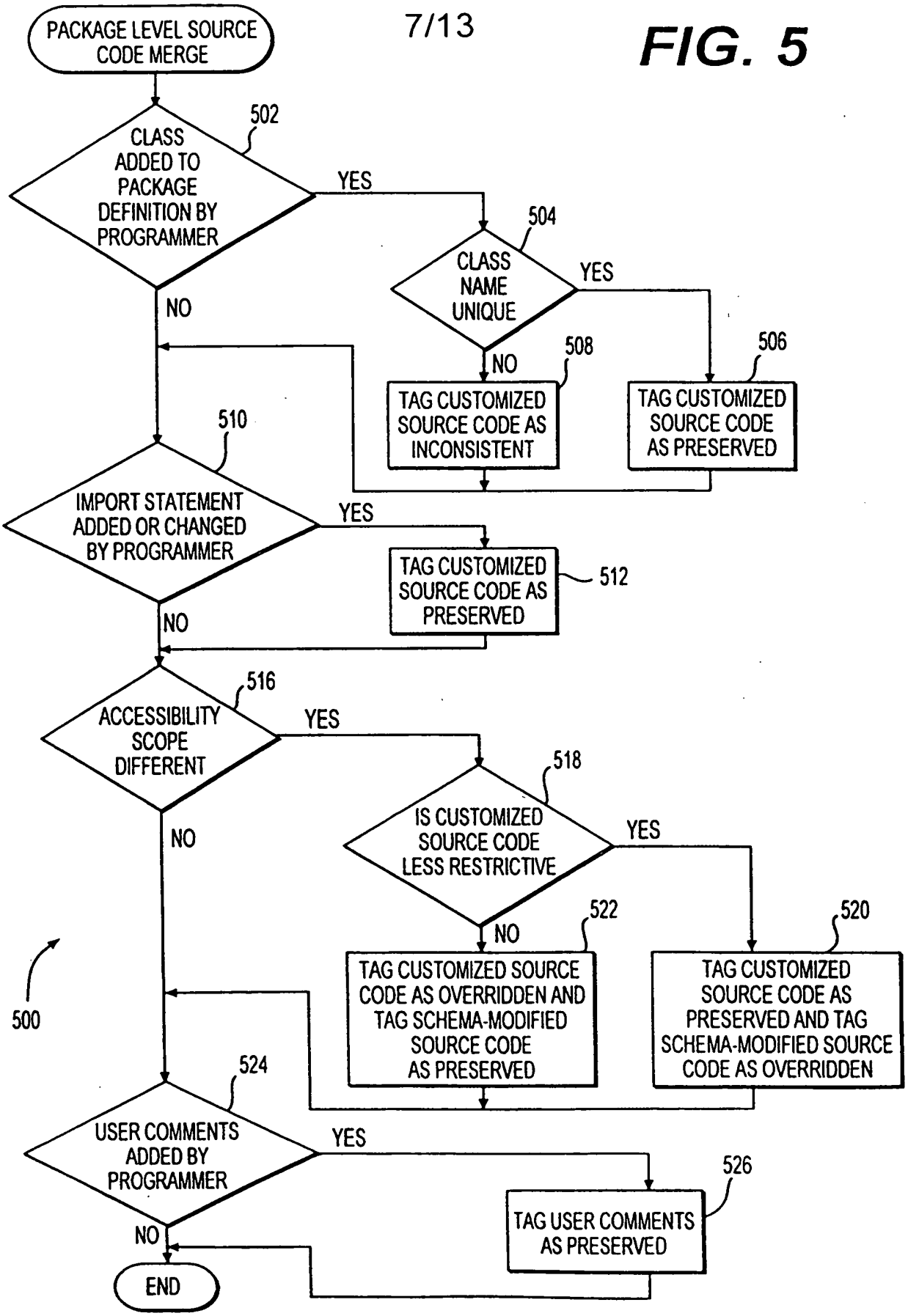


FIG. 6A

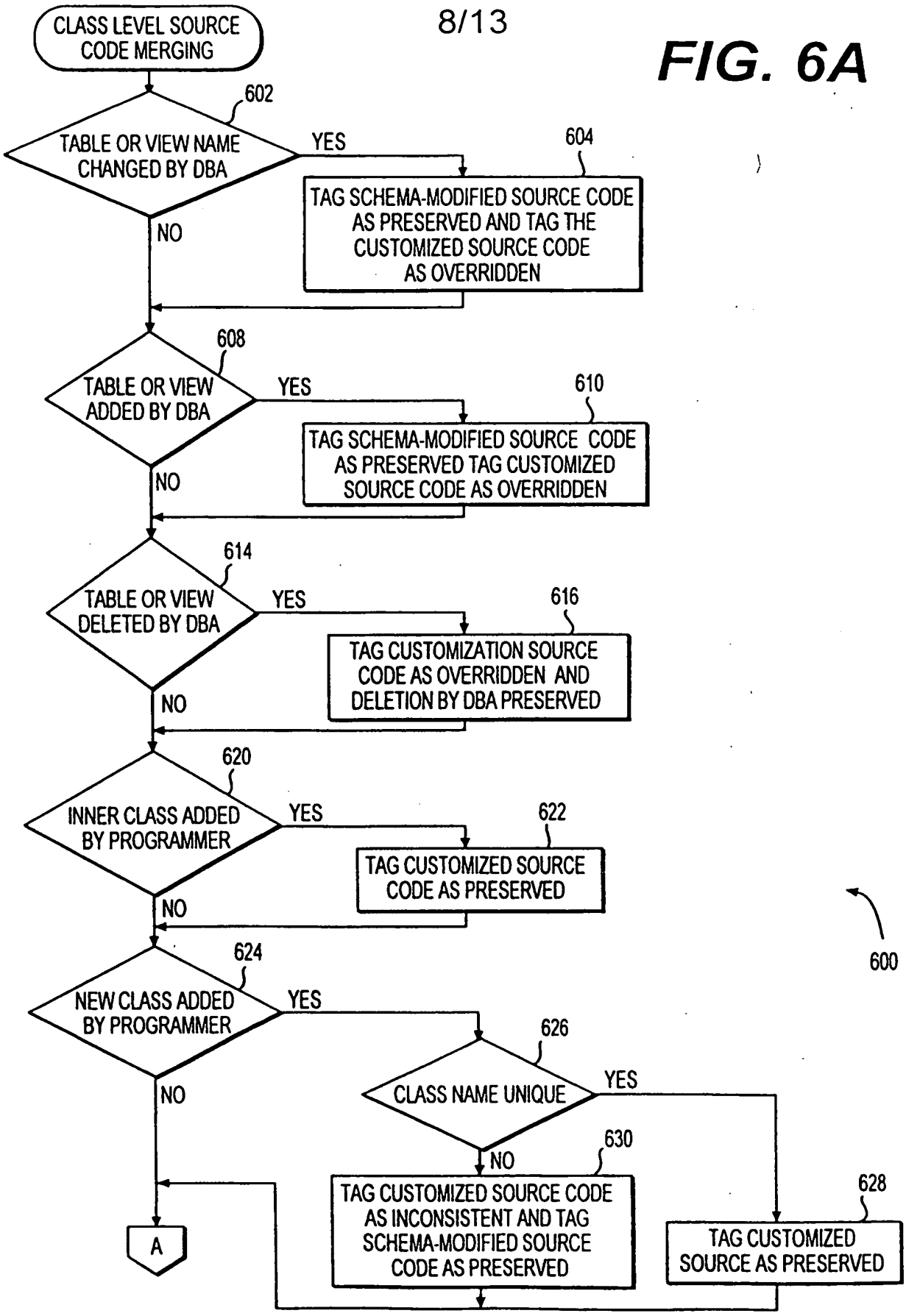


FIG. 6B

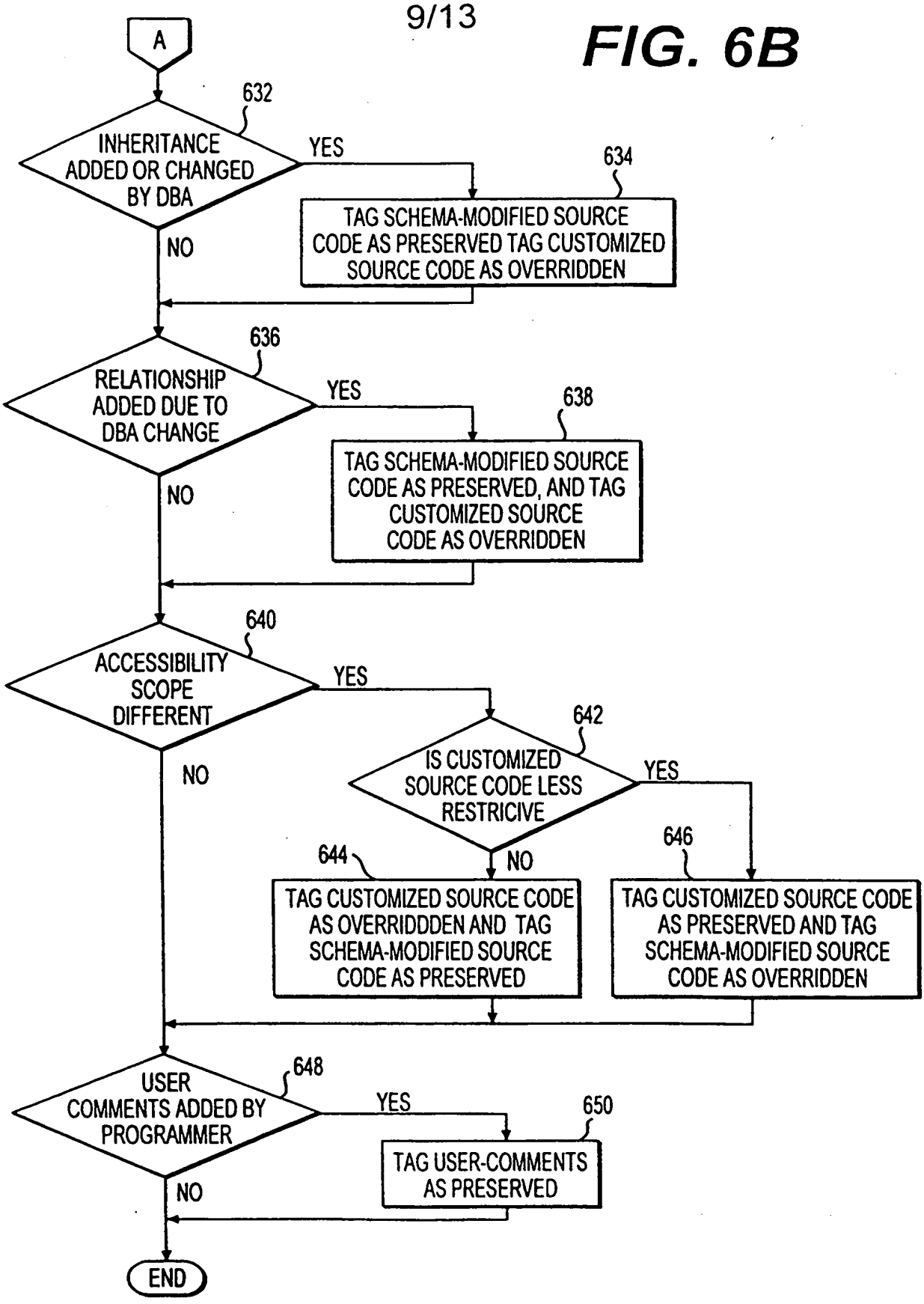
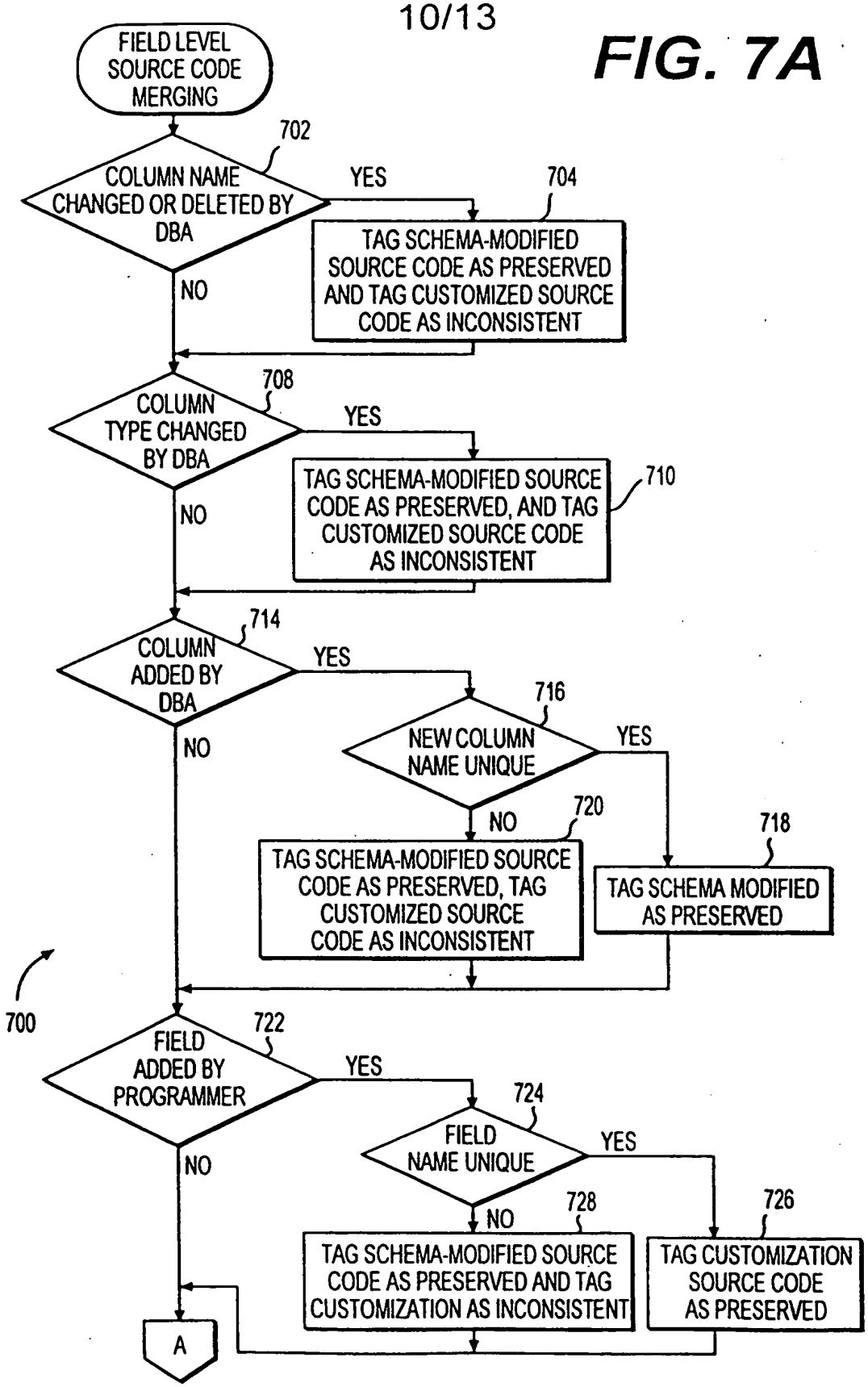


FIG. 7A



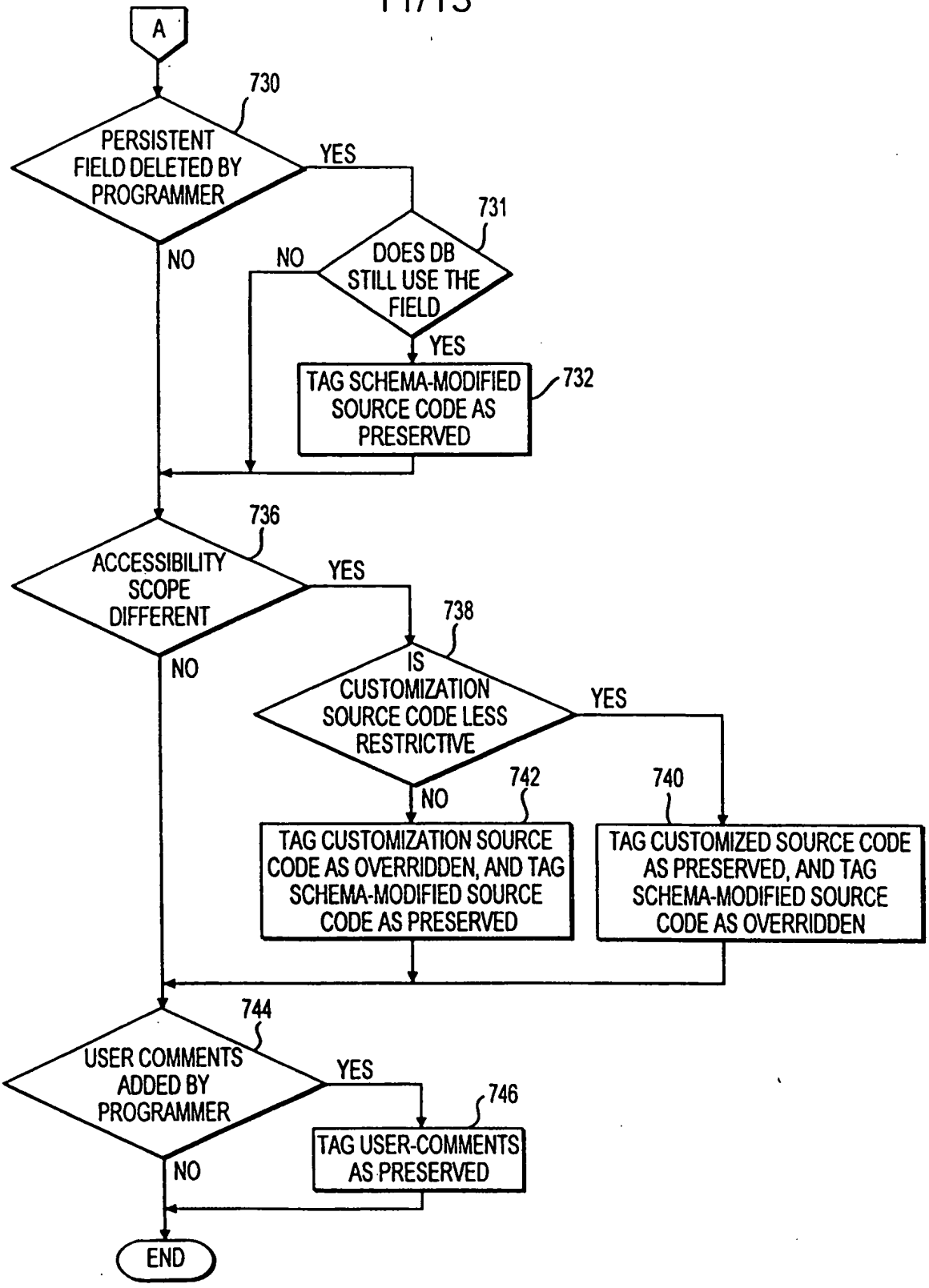


FIG. 7B

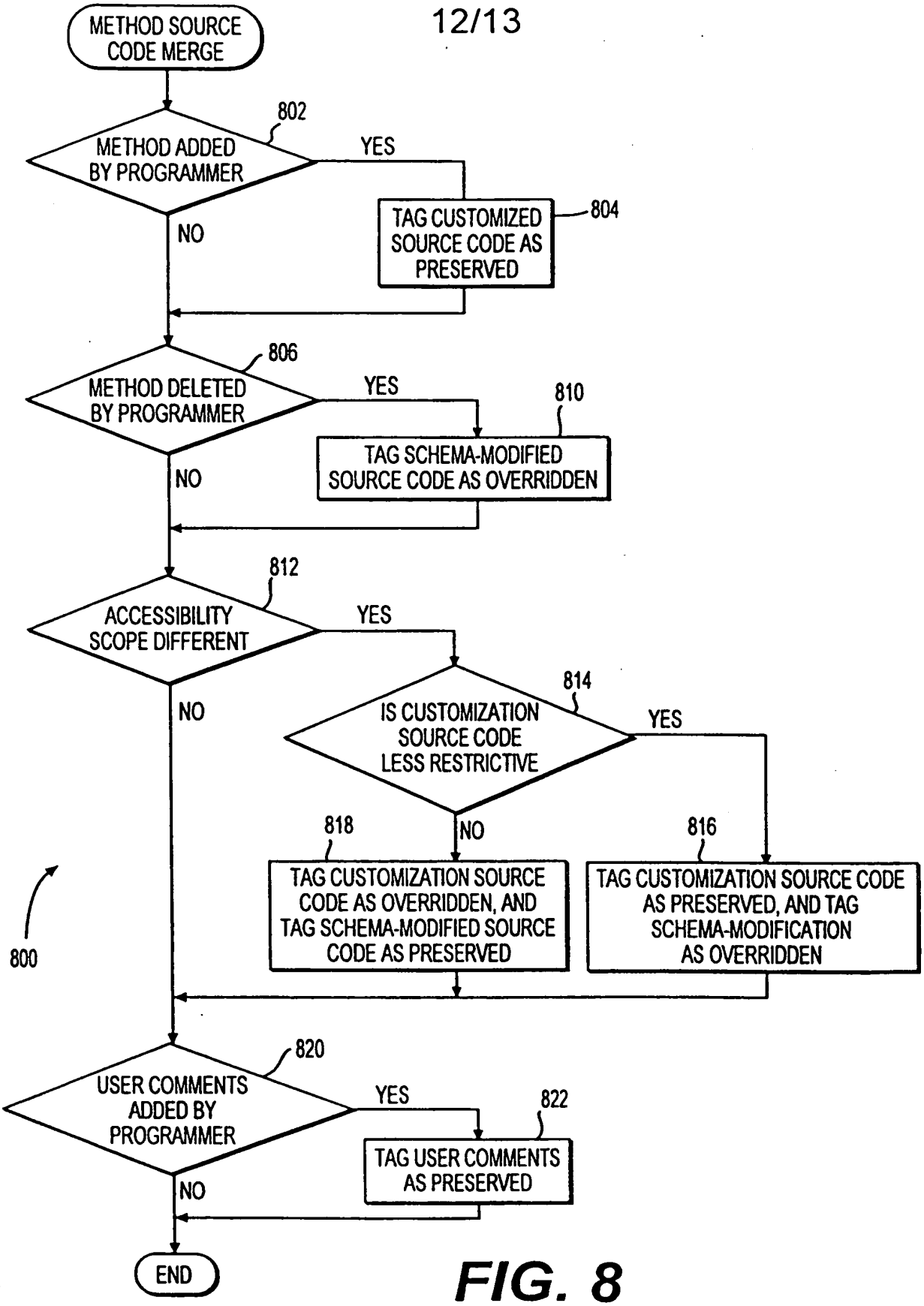


FIG. 8

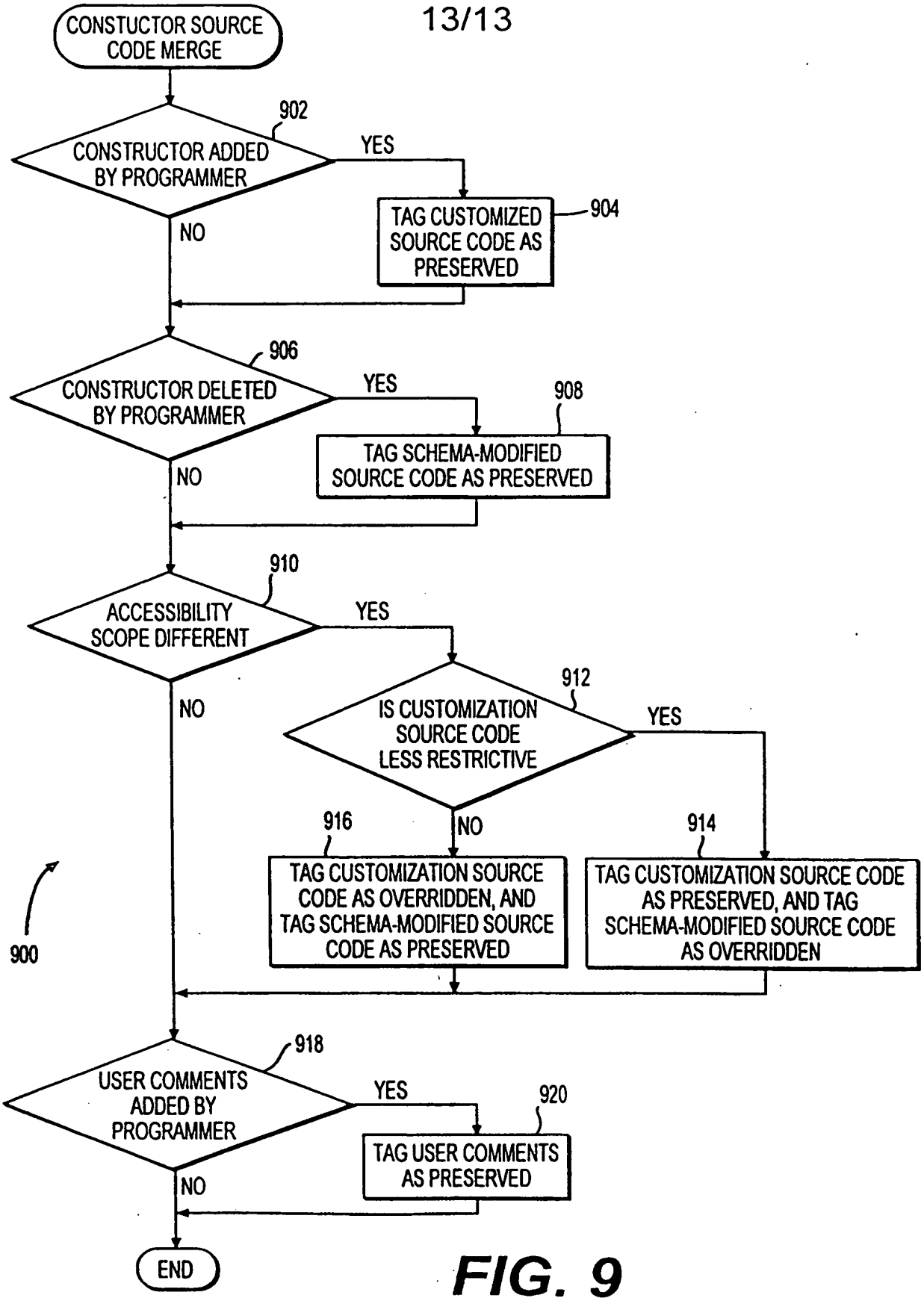


FIG. 9

INTERNATIONAL SEARCH REPORT

Internat .al Application No
PCT/US 98/27247

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F17/30 G06F9/44

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	"A White Paper: TOPLink for Java" 1997 , THE OBJECT PEOPLE , CANADA XP002098153 see page 3 ---	1
X	"AUTOMATICALLY REVISING FUNCTION PROTOTYPES IN C ANC C + + IMPLEMENTATIONS OF SYSTEM OBJECT MODEL CLASSES" IBM TECHNICAL DISCLOSURE BULLETIN, vol. 37, no. 8, 1 August 1994, pages 363-365, XP000456452	2
Y	see the whole document --- -/--	1

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
26 March 1999	13/04/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl. Fax: (+31-70) 340-3016	Fournier, C

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Intern. al Application No
PCT/US 98/27247

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	"Sun simplifies database programming with Java Blend" PRESS RELEASE / SUN MICROSYSTEMS, MOUNTAIN VIEW, CA, USA,21 August 1997, pages 1-3, XP002098152 http://java.sun.com:8081/pr/1997/august/pr970821-01.html see the whole document ---	1,2
A	EP 0 472 812 A (LANDIS & GYR BETRIEBS AG) 4 March 1992 see abstract; claim 1 -----	2

1

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Intern. ial Application No
PCT/US 98/27247

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0472812 A	04-03-1992	DE 59108978 D	10-06-1998

Form PCT/ISA/210 (patent family annex) (July 1992)